IEEE International Workshop on Intelligent Robots and Systems
IROS '90

The Evolutionary Design of MCPL the MSS Command and Programming Language

V. Hayward, L. K. Daneshmend,
A. Foisy, M. Boyer, L.P. Demers,

McGill Research Center for Intelligent Machines
McGill University, Montréal, Quebec Canada H3A 3A7

R. Ravindran, T. Ng

Spar Aerospace Limited, Remote Manipulator Systems
1700 Ormond Drive, Weston, Ontario Canada M9L 2W7

Abstract: The remote manipulator system designed by the acronym MSS, which Canada is contributing to the International Space Station, is briefly described. The underlying structure of MSS is analyzed in terms of a collection of hierarchies. Control language design issues are then analyzed and an object-oriented methodology is proposed with a view to define a run-time structure in relation with task planning requirements.

Key words: Robotics, Remote Manipulation, Space Exploration, Robot programming, Object-oriented software.

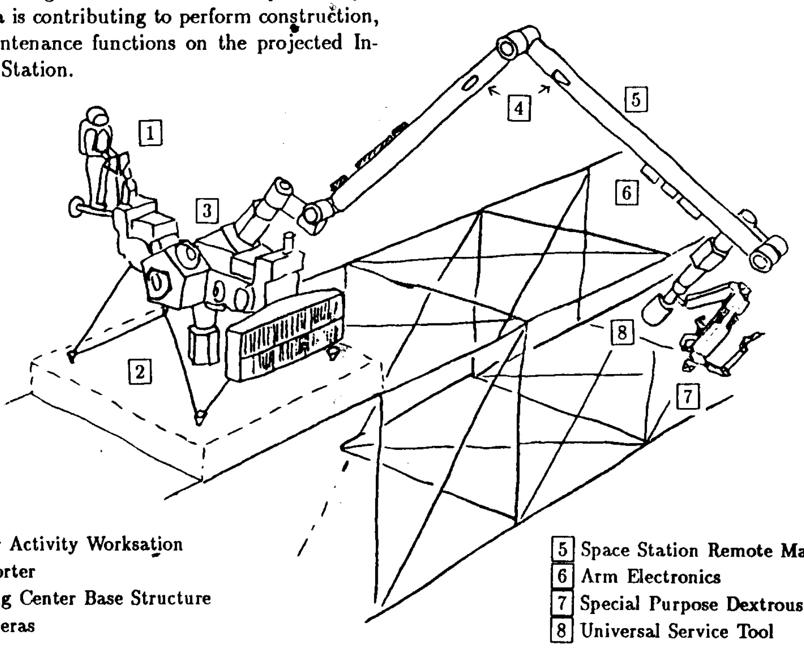
1 Introduction

The MSS acronym designates the remote manipulator system which Canada is contributing to perform construction, servicing and maintenance functions on the projected International Space Station.

A language called MCPL (MSS Command and Programming Language) is being designed to enable advanced operator and computer control of MSS in a variety of tasks. The design strategy is based on an evolutionary development since the MSS is a long lived project of the order of thirty years, spanning multiple steps of enhancement.

The requirements for the space operation of manipulators are quite peculiar since the manipulators are meant to operate under extreme conditions in addition to stringent reliability requirements. The complexity of space operations require a degree of versatility rarely encountered in known robotic systems.

Canada has acquired a unique expertise for the development of such manipulators since Spar Aerospace Ltd. has designed and built the Canadarm, the Space Shuttle Remote Manipulator System.



1	Extra-vehicular Activity Worksation
2	Mobile Transporter
3	Mobile Servicing Center Base Structure
4	Television Cameras
5	Space Station Remote Manipulator System
6	Arm Electronics
7	Special Purpose Dextrous Manipulator
8	Universal Service Tool

Figure 1: Mobile Servicing Center

2 A Short Description of MSS

MSS stands for "Mobile Servicing System". It involves a collection devices, the most notable of which are: (1) Slave elements: Power Management, Computing Resources, etc.; (2) Passive elements: maintenance depot, tool rack, etc.; (3) Manipulator and attachements space station manipulator, two special purpose dextrous manipulators, effectors; (4) Person-Machine Interfaces: IVA control station, EVA workstation, etc...

MSS on-orbit requirements describe what is conventionally called a "telerobot", that is a manipulation system which is capable of functioning under human operator supervision, autonomous control, or in shared mode.

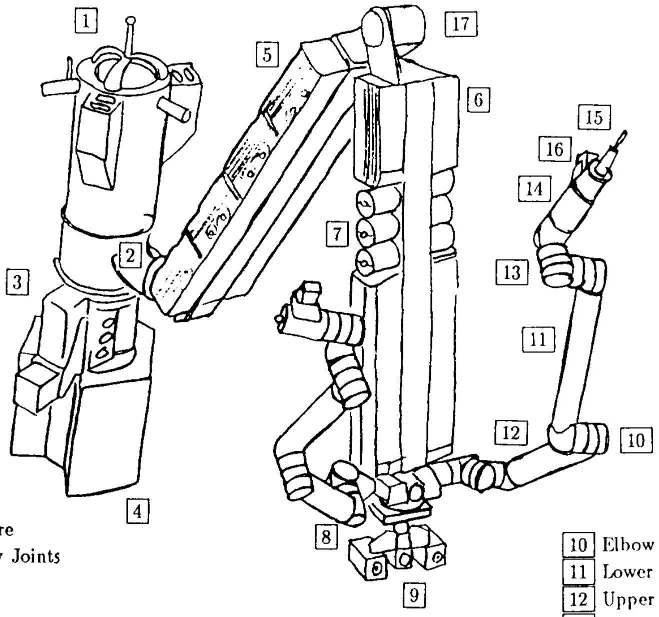
From the manipulation view-point, MSS is composed of two main elements. The SSRMS or "Space Station Remote Manipulator System" is a large seven degrees of freedom manipulator meant to provide a large load handling capacity and considerable reach. The SSRMS is equipped at both ends with a latching end-effector designed to mate with grapple fixtures situated at all relevant places of the MSS and the Space Station structure (see figure 1). The SSRMS is normally affixed to the "Mobile Servicing Center", itself attached to a mobile transporter that can travel along the Space Station structure.

The SPDM (see figure 2) is primarily intended to be operated from the end of the SSRMS. It can also be operated through its latching end effector from any of the grapple fixtures located on the MSC, the MSC "Maintenance Depot", or on the Space Station truss.

The reader is referred to [5] for additional details on the design of the system.

3 Structures

A robotic system such as MSS can be better described from various perspectives, each leading to a different hierarchy [3]. This decomposition should form the basis of the design of the language meant to control such a system. Since a language is defined by its syntax and its semantics, we found appropriate to be concerned in a first step by its semantics rather than by its syntax. Thus, our study provides a conceptual architecture concerned with procedures involving effectors, sensors, and multi-level plan interpre-



1 Power Data Grapple Fixture
2 Lower Body Yaw and Body Joints
3 TV Camera
4 Latching End Effector
5 Lower Body Electronic Boxes
6 Upper Body Electronic Boxes
7 Upper Body Tools
8 Shoulder Roll Yaw and Pitch Joints
9 Stereo Camera and Light

10 Elbow Pitch Joint
11 Lower Arm
12 Upper Arm
13 Wrist Pitch Yaw and Roll Joints
14 Force Moment Sensor
15 Special Tool
16 TV Camera
17 Upper/Lower Body Pitch Joint

Figure 2: Special Purpose Dexterous Manipulator

tation. However, the actual *form* of the commands, which is a person-machine communication issue, is not discussed because it falls outside the scope of this paper.

The decomposition that we suggest here is by no means unique and may vary from one telerobotic system to another. We believe however that the multiplicity of hierarchies is a property shared by most complex telerobotic systems which is vividly illustrated by the MSS project. Let us discuss some of these proposed hierarchies.

3.1 Activity Sites

All telerobotic systems are in essence distributed systems, in the physical sense and in the computational sense. Of course MSS is no exception. At least three physical sites may be distinguished in terms of activity.

The local site, situated on earth, in a laboratory environment, may possess facilities which are limited only by available technology. It may include computer based simulators as well as physical mock-ups of the remote site, as it is often the case in space exploration. This site may be equipped with powerful person machine interfaces offering any required modality: graphic, kinesthetic, linguistic (written and speech), and so-on.

The operator site is located near the operational site, often within the direct visual range of a human operator. In the case of MSS, this site may be situated either within pressurized head-quarters (Intra Vehicular Activity or IVA) or at the base of the manipulation system (Extra Vehicular Activity or EVA). The requirements imposed by the environmental conditions restrict considerably the range of possible person-machine interface modalities. This site might be defined by the physical reach of a human operator during normal operation.

The remote site is defined by the manipulation system and where it can reach. As described above, it comprises the SSRMS (main manipulator) and two SPDM's or dextrous manipulators and their working envelop.

Clearly, these sites form a hierarchy of decreasing richness of capabilities.

3.2 Computational Sites

Distributed computations are required over the entire system. On the local site, computational and storage capabilities can be as large as required. On the operator site, power and space limitation put severe constraints on the computational and storage capacities.

Each site forms a computational node of limited capacity. Each of these nodes are linked by communication channels which also are limited in capacity and speed. For example the earth-station link causes inherent communication delays and has a limited bandwidth. On the space station, communication delays can be made much shorter and bandwidth higher. At the remote site, in the manipulator system itself, communication delays have to be made very short due to the necessity to perform feed-back control. Thus, we can observe a computational hierarchy with nodes of decreasing computational power, and channels of increasing rates, and decreasing communication delays. In the details, the computing hierarchy is more complex than the one just outlined.

3.3 Sensing

There is clearly a physical layer. In the case of sensors it can be called the instrumentation layer. In some case, sensor data can be utilized almost directly, for example, in a tracking task. Most often, raw data will require some kind of filtering, but still will represent the sensed variables. Filtering only requires knowledge about the signal itself. One level up, data needs to be aggregated, possibly from different sources or through time, in order to construct models. Aggregation require knowledge about the properties of the sensor. Finally, aggregated data will require interpretation, that is, one will have to account for the nature of what is being sensed in order to derive the require information. Interpretation also requires knowledge about properties of what is being sensed.

In short, there are apparently four layers of sensing hierarchy, which will not necessarily match layers of the others hierarchies. This sensor organization must be apparent in the data-base that specifies available sensing capabilities (figure 3). The information follows an ascending pathway. There are of course many issues not mentioned in such a short account of the sensing process. However, this simplified picture should suffice for the discussion at hand.

Interpretation Layer	*Knowledge about the world*
Aggregation Layer	*Knowledge about sensors*
Filtering Layer	*Knowledge about the signal*
Instrumentation Layer	*Knowledge about Physics*

Figure 3: Sensing Layers

3.4 Commands

This hierarchy is the most often discussed. It relies on the assumption that any command (descending signal) can be decomposed into other commands, hopefully simpler and concerning lower levels of abstraction. A command is normally specified in terms of constraints (arguments) applied to a labeled method (algorithm).

A five layer functional hierarchy has been suggested by space operation specialists. For example, the top layer, or function layer, corresponds to types of missions: `Space-Statio` `Assembly`, `Transportation`, `Pay-load Handling`, `Pay-loa` `Servicing`, `Space-Station Maintenance`, `EVA Support`, `Safe-haven Support`. The next layer encompasses operations. For example: `Assembly of Modules` in `Space-Station Assembly`, `Fuel Tanks` in the `Transportation Function`, `Clean Windows` in the `Maintenance Function`, and so-on.

The next layer is the activity layer, for example, `Track` and `Capture the Orbiter` in a `Berthing Operation`. The next layer is a task layer, for example, `Capture the Orbiter` for the activity just mentioned. Finally a sub-task layer deals with the effector and arm control. For example, `Move to Location`, `Latch`, and so-on.

In a robotic system such as the MSS, the bottom layers may correspond to robot programs which encapsulate basic operations in terms of strings of gross motions, and sensor-based fine motions sequences (guarded and compliant) together with the operation of end-effector and peripheral equipment [4]. The next layer will be concerned with trajectories, for which the mechanical system is abstracted in terms of points in velocity/force space. Finally, the lowest layer will consist of dynamic control algorithms applied to explicit set-points, whether they concern the motion of manipulators, end-effectors, or peripheral equipment.

A tentative run time structure on which this command hierarchy could be mapped is presented in a subsequent section, while levels of abstraction are depicted below.

AI Planner	*Predicates and Logic*
Strategy	*Skeletons*
Actions	*State changes relevant to the plan*
Motions	*End point motions. Homing, tracking*
Control	*Tool velocity and force*
Joints	*Angles and torques*

Figure 4: Possible Command Hierarchy

3.5 Motion Planning Hierarchy

Motion planning can be viewed as a process occurring between the task planning process and the servo-control process. The role of motion planning is to satisfy a set of constraints dictated by the manipulator itself (its work envelope, kinematic and dynamic properties, and possibly other considerations such as deflection), the task (nominal trajectories must converge toward a goal state under model uncertainty), the environment (motions must only generate wanted collisions with controlled force), and design parameters such as energy or joint travel minimization.

Many of the motion planning techniques require extensive computations. In consequence, the very first stages of task planning consist of deciding how much motion planning must be done off-site, at task preparation time, and how much can be done on-site. On-site planning leads to a larger amount of flexibility and adaptation from the system. In the "programming by showing" systems, all trajectories are stored in a fixed manner. In sensor-based motions, reference coordinates and target positions can be determined at run time. The dynamics of a manipulator can be utilized on-line to set bounds on accelerations. However, in sensor-based motions, one must insure that the resulting trajectories are collision-free. This check cannot of course be performed off-line. Thus, it is a desirable goal to include that capability at run-time since it would increase the adaptivity of the system to new situations.

There also exist a classification in the nature of motions. Gross motions are utilized to move manipulator and loads over large distances. In this case, the principal constraint imposed onto the motion is avoidance of collisions. In the case of docking, for example, there exist additional constraints such as following a well defined path in Cartesian space. Fine motions will be used to reduce the discrepancy between expected model-based trajectories and actual trajectories constrained either by physical contact or proximity sensing.

3.6 Physical Hierarchy

The structure of the MSS project suggests a hierarchy in the very mechanical design of the system. Whenever the transporter moves, the entire system moves. Consequently, transporter motions are, from a geometrical point of view, affecting all parts of the system. The same can be said from the SSRMS, from the body supporting the SPDM's, docking module, etc... However and fortunately, structural constraints forbid arbitrary combinations of motions to occur.

The very physical structure of the system also suggest several hierarchies in terms of reach, load capacity and dexterity.

3.7 Granularity of Description

There also exists a hierarchy in the abstractions and models used to describe the system. At the highest level, the system and its task can be described in terms of formal logic, once hardware details are abstracted. The language implementor has to choose the level of abstraction or *grain size* at which such a description is appropriate. Although in theory, mathematical logic is the only tool required to describe any kind of system, in practice, one must decide when this approach becomes appropriate. At a less high level of abstraction, it might be useful to describe the system in terms of automata. In other words, the system is described in terms of *state changes*. Some formalisms such as Petri Nets have been proposed to express concurrency. At an even lower level, the system can be described in terms of *processes*. Processes have a finite life time and explicitly deal with the notion of time, hence the importance of synchronization mechanisms. At a less high level, descriptions are made in terms of continuous functions (i.e. kinematics) and continuous feed-back control. But again, strict hierarchical order is hard to enforce. For example, a single plan might involve continuous notion such as "`Track(Surface)`" or discontinuous ones such as the predicate "`Collision-Free(Trajectory, Environment)`".

3.8 Rates

The rate of flow of information is an important criterion of classification.

The rate hierarchy will be determined by the capacity and delay of the various communication channels. It is particularly important to quantify these parameters because they determine the nature and the usefulness of the possible intervention of the human operator. For example, because of transmission delays, kinesthetic coupling will be avoided between the ground site and the operation site. Clearly, the various control layers operate at various rates and so do the sensory layers. Rate compatibility must be enforced.

Transmission delays can be overcome in certain cases using preview control in the case of autonomous control, using predictive displays in case of teleoperation.

The hierarchy has been placed last in our discussion because we now suggest the reader to mentally perform a mapping of the rate hierarchy onto each of the others described before. Thus, it is easy to become convinced of the irregular and possibly time and task-varying nature of the underlying structure of the system. Of course, this exercise ought to be performed with all possible combinations to get a feel of the complexity of the problem.

4 Run Time

The run time structure should account for the existence of multiple hierarchies in a robotic system such as the MSS. In a first step, we have suggested a run-time hierarchy made of the following triplet: verification, execution, and observation (VEO) as a basic building block. Refer to the following subsections for a short description of each component of the triplet. This run-time hierarchy will be composed of n levels and level 0 will be known as the physical level. It is at the physical level, which corresponds to the instrumentation layer, that actions are physically performed and that the sensors interact with the world. At the top level, a high level plan is given and that plan is refined in the subsequent levels until the physical level is reached.

At any given level, a plan is verified and the execution of the plan is carried out. The result of the execution returns information which is aggregated in observations. The first level of information comes from the sensors. For other levels observations come from aggregation of other observations.

At any given level, the VEO triplet is autonomous which gives a good abstraction by restricting the exchange between levels only through the execution/verification interface. Furthermore, execution of actions goes from the top level to the physical level and the aggregation of information flows from the physical level to the top level. This flow of information resembles the NASREM model [6].

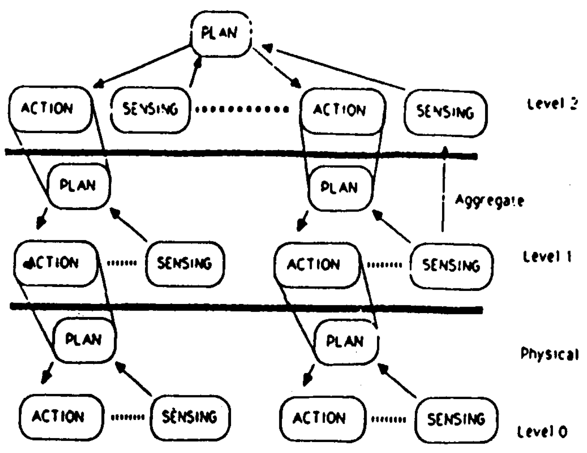


Figure 5: Structural hierarchy of system: The heavy line is a frontier between levels, the groupings emphasize the interface between levels.

4.1 Verifying

Plans are givens to the system and are provided to each level of the run-time structure. A step of the plan will be checked using available information about the current context, and actions will be executed. A plan step will give rise to an outcome which is a context change or *side effect*. This leads to the expansion of a plan step into actions. Each action of the current level represents a plan at the following level, which gives rise to a recursive definition of the interpreter.

The actual outcome is sensed and checked against the expected outcome. The *method* has to make the decision whether the current plan needs correction at the current level of interpretation (i.e. compare sensed variables against thresholds).

4.2 Executing

Execution of actions results from the interpretation of a plan. We must distinguish between two kinds of actions: real actions and virtual actions. Real actions are those that take place at the physical level. These are the outcome of the upper levels. Virtual actions are those generated by the interpretation of the plan at different levels other than the physical level.

4.3 Observing

Observations originate from the aggregation of information coming from other observations or coming from the sensors. Observations are fed back in the checking of the plan to help verify the outcome of the actions or simply satisfy other preconditions that will produce other actions. Notice that aggregation of information implies that forms of sensor fusion may be central to the system.

4.4 Exception and Sensing

The decision mechanism handling the outcome of actions might be thought of as an exception mechanism. According to the outcome of each action, three cases must be envisaged. Either the outcome is within expected bound and the plan may be carried out without further processing, either the outcome requires a modification of the next step or the choice of another control thread of the same plan (an error), or the outcome requires the abandon of the current plan and control is transferred at a higher hierarchical level. For example consider the case of a guarded motion. Either the sensor predicate does not indicate the presence of an obstacle and motion continues, either the sensor predicate indicates the presence of an obstacle and a different move command must be issued. Finally, if the sensor predicate does not indicate the presence of an obstacle and the robot is driven into a singular or joint limit configuration, then the current plan must be abandoned and failure reported at a higher level. It is really an exceptional case since a higher plan level should have made sure that the obstacle was indeed reachable.

The handling of these situations can be analyzed in some cases but this remains a difficult problem in general. We suggest the reader to refer to [1] for a more thorough discussion of this topic.

5 Language Issues

In a robotic context, a *language* serves three purposes: (1) Express the task that the robot is to perform at one or several levels of abstraction; (2) Implement some or all of the control algorithms, including representations attendant to the task, the robot, and the environment; (3) serve as a medium of communication between a user/operator and the machine. Clear design advantages can be drawn from using a single language to fulfill all these requirements.

The hierarchy of commands has to execute on a physical and computational structure, consequently, a mapping is established between the command or *operational* hierarchy and the *functional* hierarchies which describes the structure of the system. We call this process a *binding* process.

If the functional hierarchy can directly execute the abstract task description, this binding step may be omitted and everything is interpreted. This is usually the case in "industrial robotics." Most of the binding is decided at factory design time. Machines and robots are assigned very small and fragmented tasks. In that case planning is reduced to scheduling. Furthermore, the factory design enforces matching between levels of abstraction and functional layers. This is in fact the case in many large scale technological systems which all attempt to enforce a hierarchical structure. For example, power distribution networks are designed in such a way that local failures do not disturb the function of the whole: normally a short-circuit in one house does not affect the neighboring one: local re-planning is easy.

Robot control systems can be molded into the hierarchical framework if the variability of tasks to be performed is small. Unfortunately, in applications such as space automation, the considered class of tasks is very broad. The task itself is what is controlled, leading to a great deal of variability. For each task, we obtain a different physical system, thus potentially different levels to describe it.

If the binding is performed at run time within levels, we obtain a more complex but more versatile system. This is the case when the execution Finally if the entire binding is done at run time, we obtain a very complex system, but immeasurably more versatile (probably what biological systems do).

The type of plan interpreter proposed here is indeed a very simple one because each plan step is only made of possibly conditional actions, which in turn may be considered as plans. The only escape mechanism being the handling of exceptions.

One extra step in generality would involve a *plan interpreter* which would *dynamically* bind actions to plan steps according to the context. For example, if a plan step involves unfastening the panel of an ORU to be serviced, the initial plan may not specify which one of the two *spdm*'s would be allocated to execute this plan step. The decision can be made at run time according to the context. Of course, such a strategy places much heavier capability requirements on the task interpreter. We feel that such dynamic binding should be limited to well suited cases but *must* be accounted for.

6 Discussion

As just discussed, supervised/autonomous robotic systems may be characterized by software environments which are very large, long lived, have functional requirements which may evolve, and which deal with a large number of mechanical and computer subsystems, which are distributed physically, which function in parallel temporally, and may also evolve.

There is no inherent reason why a complex robotic system should prove itself amenable to imposition of a rigid taxonomy upon its structure. Rather, it is the design process itself which imposes order upon the system, such that the structure can remain invariant throughout various task executions or operations.

Strictly hierarchical designs of robotic systems, which rigidly enforce simple inheritance, have been notoriously unsuccessful in the past. Since the operation of such a system may be viewed from so many differing perspectives, simple inheritance is much too restrictive a representation tool.

We have proposed to base MCPL on an object-oriented paradigm for the implementation of the run-time structure and of the databases that support it. This section will motivate the use of object-oriented systems, present a brief analysis of them, their relevance to telerobotics in general, and to the MCPL project in particular.

We would like to put forward the following motivations: In the development of a software project the design phase results in a description that must be easily implemented, provides all required functions, and easily maintained. Going to the implementation phase all the decisions should have been made. The job is now to take the detailed design and transform it into code, and check if the code implements the design.

The values of object-oriented "programming" are that it touches both the design and implementation phases. In fact, it tends to blur the distinction between the two phases. This means that you can write a skeleton of the system by identifying the relationships between objects and then flesh out their functionality. This gives rise to early prototypes since it is all done with the same programming environment: the object-oriented one.

A more concrete grasp of the object-oriented paradigm can be gained by viewing it as a means of achieving modular software resource management. Classes and objects are the modular building blocks. Inheritance allows classes to specified in a modular, incremental, manner. Since objects are first-class computational values, i.e. they may be assigned as values of variables or passed as parameters, they can be managed directly by computation within the language.

From a practical point of view, classification is an asset to software management because it allows objects to be managed from within the language, in the same manner as data. Classless languages (e.g. Ada) rely upon special purpose language facilities to handle module management, e.g. library facilities. Module management may be trivial for a software system with less than a 100 modules, but becomes critical as the number of modules and module types increases. Inheritance provides a means for specifying relations between classes of objects from within the language, and hence allows *evolution* of the system over time.

Thus, from a software engineering perspective, the objet-oriented paradigm appears suited to the programming of very large, long-lived, software systems. It must be remembered however, that the object-oriented approach also imposes certain additional responsibilities on the designers of a software system, in the form of very fundamental and crucial design decisions. The selection of classes, objects, and their interfaces, ultimately impacts all aspects of the performance of an object-oriented system, and is inherently application-dependent.

It appears intuitively obvious that typed (classified) universes of discourse are more expressive than untyped universes. The introduction of classes may be viewed as a means of *filtering* unstructured information. It carries with it the responsibility of choosing the appropriate level of abstraction and interpretation, if the objective of reducing complexity is to be achieved.

A hierarchical structure can be achieved if first-order classification of data is extended to second-order classification of classes. Hierarchies enable the sharing of properties of "ancestors" by "descendants", i.e. inheritance. Implementation hierarchies are concerned with incremental evolution of properties of classes over time, i.e. with the *implementation* of objects, and hence are relevant to program development. Specification hierarchies are concerned with relations between classes, in terms of abstract interface specifications, and hence are relevant to system design. Multiple inheritance enables a class to be viewed from a number of different hierarchical perspectives.

If classification is to be utilized constructively as a means of structuring information, an appropriate set of criteria for classification, i.e. a *taxonomy*, is essential. The taxonomy employed must be selected prior to the classification phase, and hence impacts the resulting hierarchies.

Another critical aspect of the MSS software environment will be its physically *distributed* nature. The object-oriented approach is an aid in promoting the modularity, via objects and message passing, required for distributed systems implementation. However, communication bandwidth between distributed components is a practical constraint which must be taken into account in the taxonomy of the object-oriented solution. In addition, inheritance requires the dynamic sharing of code: although this may be circumvented by replicating code in different distributed components, replication partially defeats the advantages of sharing and inheritance.

7 Conclusion

Software design of the programming/planning and task execution of a advanced robot system task may be viewed as a problem of defining two taxonomies and the relationships between them. Upon closer investigation of the problem, the requirements of this design problem are seen to be five-fold:

- Design of a *Task Planning* taxonomy
- Design of a *Task Execution* taxonomy
- Mapping of the robot task to the planning taxonomy
- Mapping of the robot task to the execution taxonomy
- Linking the planning and execution paradigms.

The requirements for these two taxonomies are, to a large extent, conflicting. Task planning requires an *operational* taxonomy, in which the *actions* to be performed are classified. Classification of actions is beneficial in planning since it leads to pruning of the set of operators to be considered in constructing a plan, and facilitates the construction of generic exception handling operations [1].

In contrast, the run-time environment for task execution benefits from the imposition of a *functional* taxonomy, in which the physical objects to be operated upon are classified. Functional classification provides the benefits of modular development for large, long-lived, software systems.

The hierarchical object-oriented design paradigm supports a taxonomy of objects (i.e. a functional taxonomy), but not does not cater for a taxonomy of the operations which act upon those objects. We have extended the hierarchical object-oriented paradigm by specifying that: operations on objects have an associated data type and that operation types inherit attributes from operation supertypes. We arrive at the Dual-Hierarchical Object-Oriented Design (DHOOD) paradigm. This novel technique is described in a recent publication [2].

References

[1] Boyer, M., A Knowledge-Based System for On-line Robot Error Recovery, *M.Eng. Thesis*, Department of Electrical Engineering, McGill University, July 1988.

[2] Boyer, M. Daneshmend, L. K., Hayward, V., Foisy, A. Object-oriented planning, programming, and execution of robot tasks. Submitted to OOPSLA 1990.

[3] Hayward, V. 1988. Autonomous control issues in a telerobot. *IEEE Conference on systems man and cybernetics. Workshop on Manipulators in Space.* Beijing, China. pp. 122–125.

[4] Hayward, V., Daneshmend, L. K., Hayati, S. 1989. An overview of Kali: a system to program and control cooperative manipulators. *Fourth International Conference on Advanced Robotics.* Springer Verlag, pp. 547–558.

[5] Hunter, David G. 1988. An Overview of the Space Station Special Purpose Dextrous Manipulator (SPDM). *Canadian Space Agency Technical Report*, NRCC No. 28817.

[6] Oberright, J., McCain, H., Withman, R.I. 1987. Space station flight telerobotic servicer functional requirements developments. *Proceedings of SPIE, Space Station Automation.* Vol. III. Cambridge, Ma. pp. 169–172.